From: Wally Flint
To: Microsoft ATR
Date: 12/19/01 5:02pm
Subject: antitrust issues

****
THE IDEA PROPOSED HEREIN IS THE DEVELOPMENT OF A STANDARD SET OF OPERATING SYSTEM COMPONENTS. The specification of these components is not a specification for how the components should work. Instead, it is only a specification for the nature and scope of components (what module does what), together with the interfaces for the components (how to "connect to" a component, or how to access the functionality of each component). I call this operating system the "Standard Operating System" (SOP).
****

METHODOLOGIES

Many well-known software companies (BEA, IBM, Sun, ...) compete on a level playing field to produce J2EE application servers. This "fair and competitive market" did not emerge from the mist of random free market chaotic activity. Instead, it developed on the basis of the following methodologies:

1) Modularization of Software

Dell manufactures computers from video cards, mother boards, and other electronic modules and components. Contrast this with the old way of carrying out circuit design - wiring together a bunch of resistors and transistors. With the old methodology, every electronic product was essentially "custom built". Then, electronic hardware became modularized. The integrated circuit (IC) offered complex functionality (such as an amplifier) as a modular unit. Circuit boards (like a PC's mother board or video card) offered even more complex functionality as a modular unit. If a circuit board goes bad, just replace it with a new one (as opposed to replacing the entire computer). If a cheaper video card appears on the market, companies like Dell can lower costs by changing to the new cheaper video card.

This modularization could not have developed without standards. For example, circuit boards have standard connectors that plug into standard sockets in the PC. If every video card had its own custom connector, then each PC design could use one and only one type of video card.

Just as electronic products are built from standard modules, large complex software programs may be built from standard software modules. For this to happen, the interface for accessing the functionality of that component must be defined. Standardizing a software module interface is analogous to standardizing circuit board connectors. For example, if a software module draws lines on the screen, then the line drawing functionality may possibly

be accessed by calling a "drawLine" function, a "paintLine" function, a "renderLine" function, and so on. A standard is developed by choosing one of these names, and asking all component developers to use the same name. This allows software modules to be mixed and matched for a variety of purposes (optimization of cost, speed, quality, ...), just as hardware components are mixed and matched in the design of a PC.

2) Community Process

Sun has developed a community process, called the "Java Community Process", for allowing interested parties to influence the development of a standard. (www.jcp.org)

3) Proving Compliance with a Standard

To prove compliance with a standard, a compatibility test suite is developed. A compatibility test suite is a software application that exercises the various functionalities of a software module, and verifies that the behavior that results is the same behavior as that required by the standard. The same compatibility test suite is used for all software module developers, producing a "level playing field" for competition in meeting the standard.
(http://developer.java.sun.com/developer/technicalArticles/JCPtools/)

****

IMPORTANT FEATURES OF THE METHODOLOGIES DESCRIBED ABOVE

A) These methodologies allow code to remain proprietary (unless a company elects to open source its code), yet still facilitate competition for all operating system components. They also facilitate mixing and matching components. You could run a Microsoft kernel with a windowing system from company XYZ, or visa versa. Allowing code to remain proprietary stimulates competition and investment, promotes quality, and is fair to investors.

B) Part of the difficulty in solving the anti-trust problem lies in defining where the operating system ends and software applications begin. Should an instant messenger be classified as an operating system component, or is it a software application? This issue is highly significant when trying to determine whether Microsoft is bundling its applications with its operating system, and thereby forcing consumers to purchase the applications in order to get the operating system. I call this bundling phenomenon "operating system creep". Operating system creep is the process of expanding the definition of word "operating system" for the purpose of legitimizing the practice of bundling applications with the operating system.

The above methodologies indirectly provide a solution to the problem of operating system creep. Assume the standard operating system is developed as a bunch of components, instead of as one giant blob. In this case, the

standard for a given component may change frequently while the standard is maturing. However, the standard for that component will eventually stabilize, and thereafter the standard will probably not change very often.

After a component standard has stabilized, companies that develop that component are not affected by operating system creep. That is, if company XYZ markets a component for rendering the desktop on the screen, then that component cannot be adversely affected if Microsoft bundles an instant messenger component with its version of SOP. Under the current situation (no standardized modularization), the entire operating system is pushed onto the consumer as a single giant "blob" (a single giant component), and in this case, no other company can compete to provide this giant component, because the component changes with each iteration. (For example, the giant component may include an instant messenger in one iteration, where it did not include an instant messenger in the previous iteration.) But with standardized componentization, the standard for a given operating system module eventually stabilizes, and all companies can then easily compete to implement that standard. The point is that the standard for a stabilized component cannot be affected by changing the scope of what is considered the "operating system".

C) In order to end up with a quality design, an industry consortium should develop the standardized interfaces, as well as the scope of those interfaces (should it be one big interface, or a component for screen rendering, a component for I/O, and so on? should screen rendering be one big component, or should it be broken into several sub-components?). The industry consortium could standardize components using a process similar to the java community process described above.

****

ONE ASPECT OF THE REMEDY

Suppose the Windows operating system is required to implement the SOP interfaces. In this case, if Microsoft applications (such as Microsoft Word) communicate with Windows using proprietary (non-standard) interfaces, then this effectively creates an artificial shortage of applications for competing operating systems. Looked at another way, it forces competing operating systems to implement the proprietary interfaces to become "Microsoft Word compatible", and thereby destroys the standard. Perhaps one aspect of a remedy could be requiring Microsoft applications to use ONLY the standardized interfaces.